High Performance Land Surface Modeling with a Beowulf Cluster

Y. Tian

University of Maryland, Baltimore County

yudong@umbc.edu

C. D. Peters-Lidard

NASA Goddard Space Flight Center

christa.peters@nasa.gov

S. V. Kumar

University of Maryland, Baltimore County

sujay@hsb.gsfc.nasa.gov

J. Geiger

NASA Goddard Space Flight Center

james.v.geiger.1@gsfc.nasa.gov

P. R. Houser

NASA Goddard Space Flight Center

paul.houser@nasa.gov

J. L. Eastman

University of Maryland, Baltimore County

jleastman@comcast.net

P. Dirmeyer

Center for Ocean-Land-Atmosphere Studies

dirmeyer@cola.iges.org

B. Doty

Center for Ocean-Land-Atmosphere Studies

doty@cola.iges.org

J. Adams

Center for Ocean-Land-Atmosphere Studies

jma@cola.iges.org

Abstract

We designed the Land Information System (LIS) to perform global land surface simulations at a resolution of 1-km or finer in real-time. Such an unprecedented scale and intensity in computing pose many challenges. In this article we will demonstrate some of our unique developments and implementations in high performance computing with a Beowulf Linux cluster, to meet these challenges and reach our performance goals. These contributions include a fault-tolerant job management system for the cluster, high-performance parallel I/O based on GrADS-DODS (GDS) servers with dynamic load-balancing and distributed data storage, and highly scalable data replication with peer-to-peer (P2P) technology.

I. Introduction

As human activities have extended to 83% of the Earth's land surface nowadays [7], the human society's reliance on the natural resources on the land has never been greater. Consequently, the physical and dynamical processes of the Earth's land surface, such as vegetation, fresh water distribution, energy fluxes, and flood/drought events, are exerting stronger impact on the human world. Meanwhile, the Earth's land surface interacts with the other integral components of the Earth system, including the atmosphere and the oceans, and such complex interactions further drive the constant evolution of the natural environment around us.

The advance of remote sensing technology in recent years has enabled us to monitor and measure the Earth's land surface at an unprecedented scale and frequency. For example, NASA's Moderate Resolution Imaging Spectroradiometer (MODIS) satellites are monitoring the Earth's surface every 1 to 2 days, with a spatial resolution as high as from 1 km up to 0.25 km. Such observations provide a huge volume of valuable data of the Earth's land surface properties, such as vegetation, moisture and energy fluxes.

Meanwhile, to understand the internal physical and dynamical processes of the Earth's land surface, and to analyze and assimilate the satellites observations, the science community has developed various land surface models (LSMs). These computer models simulate the physical and dynamical relations between the various land surface variables, to numerically reproduce, or even predict, the land surface conditions. The leading LSMs include, for example, the community Noah land surface model (Noah: ftp://ftp.ncep.noaa.gov/pub/gcp/ldas/noahlsm/), the NCAR Community Land Model (CLM) [2], and the Variable Infiltration Capacity model (VIC: http://hydrology.princeton.edu/research/lis/index.html).

It is computationally intensive to simulate the Earth's 150 million km^2 land surface with these models, as each model has to solve dozens of equations and to deal with a similar number of input and output variables. To keep the computing-power requirements under control, the science community has been running these land surface models with coarse spatial resolutions of about 100 km to 25 km.

However, to fully take advantage of and timely assimilate the high resolution (1 km or higher) land surface data from modern satellite remote sensing measurements, we need to run the land surface models at a comparable resolution in real-time. Such resolution and speed requirements pose a grand computing challenge, as the computing intensity, both CPU and input/output (I/O), is increased 3 to 4 orders in magnitude. It can be estimated that each of the LSMs can easily produce more than 200 GB of output data for each simulated day, and would take months to finish a day's worth of simulation with a single modern CPU.

To meet this challenge, we have developed the Land Information System (LIS) [6] (http://lis.gsfc.nasa.gov) at NASA Goddard Space Flight Center. LIS is an integrated hardware/software system for high performance land surface modeling. It incorporates and drives an ensemble of land surface models, including CLM, Noah and VIC.

The computing platform is a 200-node Beowulf cluster, designed and built with special enhancements for LIS' high throughput simulations. It gives us great flexibility to experiment and implement unconventional and innovative computing technologies to gain maximum performance, as we have total access and control of the cluster's software and hardware assembly.

Based on the cluster, we have developed an array of high-performance computing techniques and components. We have developed a job management system, which is highly efficient in managing coarse-grained parallel problems, such as LIS, on distributed memory systems, especially for Linux clusters. This system features strong fault-tolerance, optimal resource utilization and flexibility. It also integrates distributed visualization so the results can be inspected in real-time without post-processing. In addition, to overcome the I/O performance bottleneck, we implemented parallel I/O and distributed data storage, and expanded the GrADS-DODS (GDS: http://grads.iges.org/grads/gds/) client-server system to support its functions over distributed data, and to support GDS server farms with dynamic load balancing. This design increased LIS performance at least 4 times. Moreover, our novel approach in applying peer-to-peer (P2P) technology to high-performance and highly scalable data replication within our Beowulf cluster, eliminated the performance bottleneck and management complexity of the conventional client–server paradigm. The implementation of the P2P system, BitTorrent (BT: http://bitconjurer.org/bittorrent/), on our cluster enables us to replicate large amount of data to all the cluster nodes simultaneously without traffic congestion, with guaranteed data integrity and with an aggregate throughput at least 5 to 6 times more than conventional NFS based file sharing.

II. LIS Architecture Overview

We designed a highly modular system for LIS' high performance modeling functions. Figure 1 shows an overview of the LIS architecture. The central component of LIS is the LIS core, which controls the execution of the land surface models (CLM, Noah and VIC), defines the parallelization scheme, and manages data input/output.

LIS continuously takes in relevant atmospheric observational data from various data repositories on the Internet as its input, and saves the data on the local storage system after certain reformatting. This function is carried out by the data retrieval component, which essentially is a collection of FTP/HTTP data downloading scripts and data reformatting programs. LIS' output data are also saved on the local storage. A key component for managing input and output data is the GrADS-DODS (GDS) server, is an HTTP-based data server to support transparent and on-demand data serving.

The LIS Beowulf cluster acts as the engine to provide the computing power needed by the demanding LSMs. LIS has software components to manage the parallel job processing, to monitor hardware status and to manage resources, in order to ensure high reliability and availability. Details of the cluster will be described in next section.

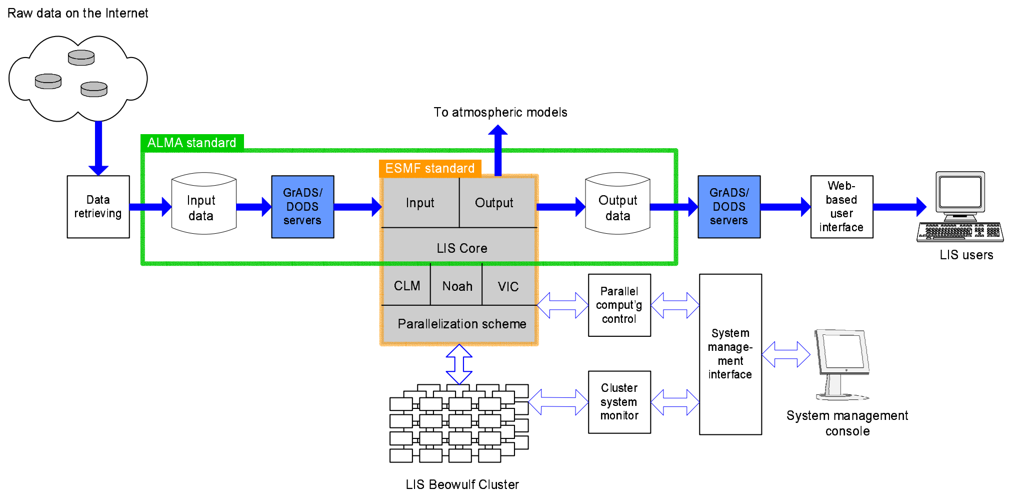

Fig. 1. Overview of LIS architecture. The central component of LIS is the LIS core, which controls the execution of the land surface models (CLM, Noah and VIC), defines the parallelization scheme, and manages data input/output. Input data are retrieved from various data repositories on the Internet (left), saved on the local storage system, and reformatted. Then the input data are served by GrADS-DODS (GDS) servers to LIS on-demand. LIS output data are also staged on GDS servers to serve users via a web-interface. LIS core and LSMs run on the LIS Beowulf Cluster by a job control system for scalable parallel computing.

An important aspect of LIS is the interoperability design by conforming to well-adopted open standards. Two major standards were followed: the ALMA (Assistance for Land-surface Modeling Activities) standard and the Earth System Modeling Framework (ESMF) [4]. The former specifies the data storage format, units, sign convention, etc., for the land surface variables so that data can be exchanged between land modeling systems easily without conversion. The latter is a generic Earth system modeling software platform which standardizes the coupling between different models. By following these open standards, LIS can be readily and easily coupled with other models, such as an atmospheric model.

In summary, LIS primarily has the following components:

- Land surface modeling. This includes the LIS core and the three land models: CLM, Noah and VIC. LIS core can be configured to run one, two or all the three land models at the same time.
- Parallel processing. Job management systems to support various parallelization schemes on the Beowulf cluster.
- GrADS-DODS server. The on-demand data serving engine to handle both input and output data. Additional performance was gained when multiple servers were running in parallel on the cluster, with proper load balancing and data replication.
- Data retrieving. It includes programs and scripts to download, store and reformat data, so the cluster can retrieve data from the Internet.
- System monitoring. It is responsible for the smooth operation of the LIS Beowulf cluster.

III. LIS Beowulf Cluster

Since the creation of the first Beowulf cluster a decade ago at NASA Goddard Space Flight Center, this technology has gained much popularity in a wide array of science and engineering applications. By connecting many PCs running open-source software, a Beowulf cluster provides a cost-effective alternative to the supercomputing facilities. The open-source nature of the software, including the operating system and most of the supporting applications, allows great flexibility for the users to experiment and develop applications that best fit their particular needs.

One of the characteristics for land surface modeling is that it is a very coarse-grained parallelization problem. In other words, the global land surface can be partitioned into pieces and each piece can be mostly processed independently without communication or synchronization with other pieces. This provides one of the best applications for Beowulf clusters, as each node can process a job independently with minimal communication with other nodes, therefore fast and expensive interconnects are not necessary for connecting the nodes.

The LIS Beowulf cluster was built with the most economical commodity components, and it has eight I/O nodes and 192 (24×8) compute nodes. Any of the I/O nodes can also act as the master nodes. Each I/O node is configured with dual AMD XP 2000 CPUs, 2 GB DDR memory, 220 GB internal disk storage, 2 built-in Ultra-SCSI channels, 2 built-in fast Ethernet adapters, and 1 gigabit Ethernet adapter (two of the I/O nodes have two each). In addition, each I/O node has one external Promise RAID systems for file storage, with a capacity of 1.2 TB in each RAID system.

Each of the 192 compute nodes has an AMD XP 1800 CPU, 512 MB memory, an internal IDE hard drive of 81 GB, and a built-in fast Ethernet adapter. Subsequent updates added another 512 MB memory to about half of the compute nodes, making the cluster a somewhat heterogeneous system.

Overall, the LIS cluster has 208 AMD XP processors of 1.53 GHz and above, 160 GB of memory, 24 TB of disk space, 192 fast Ethernet connections, and 10 gigabit Ethernet connections. The physical architecture and network layout is shown in Fig. 2. The 192 compute nodes and 8 I/O nodes are grouped into 8 branches, with each branch logically having 24 compute nodes and 1 I/O node connected to a common Ethernet switch, with all the switches stacked and inter-connected. Thus the system can be used as one big cluster with 192 nodes, or 8 small clusters each with 24 nodes, or any combination in between.

A wide range of software was installed on the cluster, including the RedHat distribution of Linux with kernel versions of 2.4.x, Sun Grid Engine (SGE: http://gridengine.sunsource.net/), MPICH [3], various compilers, network and host monitoring utilities, and so on. Detailed information can be found at LIS' website (http://lis.gsfc.nasa.gov).

IV. Job Partition and LIS Job Management System

At 1-km (1/100 degree) resolution, there are 540 million grid points on the LIS global domain (latitude: 60S 90N; longitude: 180W 180E). It is estimated that we need at least 1.6 TB memory to run such a big domain on a single computer. To use the cluster for the simulation work with parallel processing, we need to partition the job in such a way that the coarse-grained nature of the simulation can be effectively exploited.

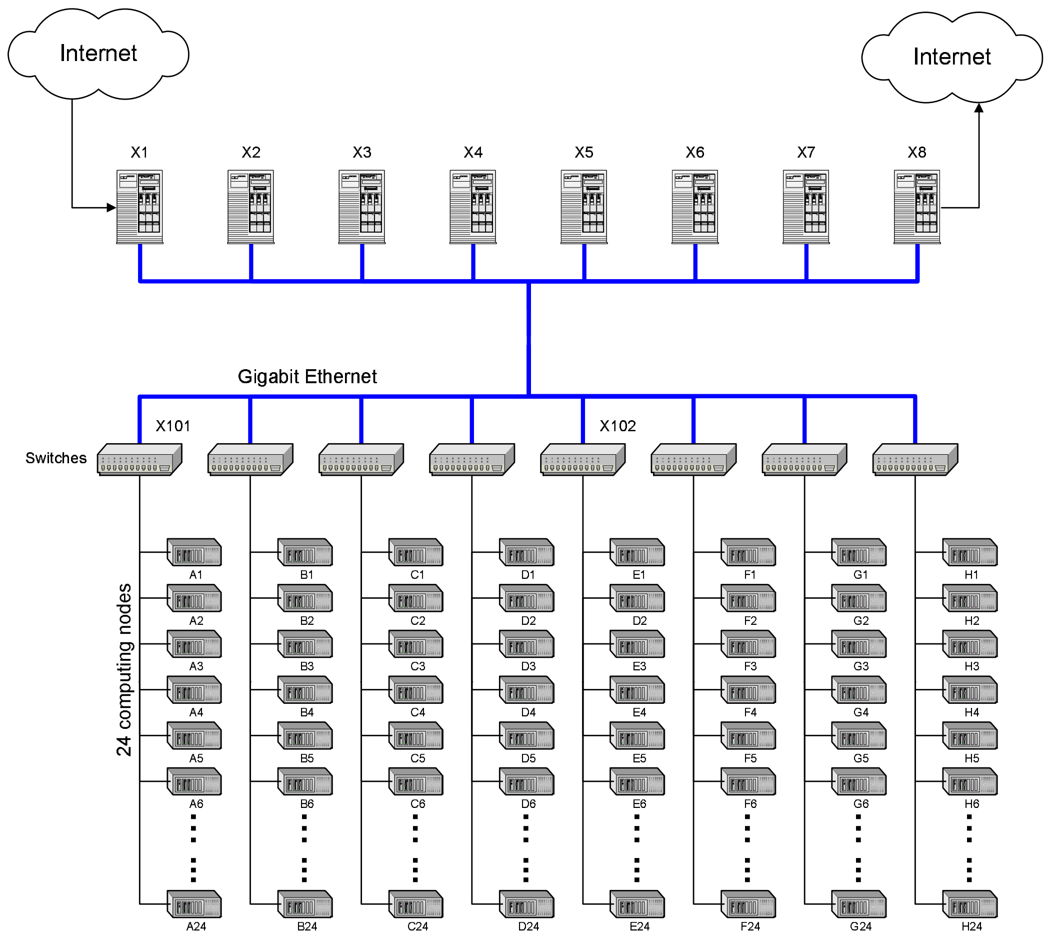

Fig. 2. Physical architecture and network layout of the LIS Linux cluster. The 8 I/O nodes' hostnames are X1, X2, ..., X8, respectively, while each compute node is named A1, A2, ..., A24; B1, B2, ..., B24; ..., H1, H2, ..., H24, depending on which of the 8 sub-clusters it belongs.

Figure 3 shows the way we split the global domain into smaller sub-domains, or blocks. An optimal size of block was selected so that each block can fit nicely in a compute node's memory. An extra benefit from the sub-division of the global domain is that we can discard all those blocks which contain only ocean points. This reduced the total number of blocks to be computed by more than half.

As each block can now be computed independently by a compute node on the cluster, we need a job management system to orchestrate the compute nodes to process the pile of jobs with scalability, performance and resource utilization. An additional critical requirement for the job management system is fault-tolerance, as failures from node crashes are not rare events with a cluster built from commodity components. The job management system need to ensure that the overall processing will not be interrupted or corrupted by individual node failures. Most of the existing job management systems, such as Sun Grid Engine (SGE), do not have a built-in mechanism to handle such failures.

We developed a job management system on the cluster based on the "pool of tasks" scheme [5]. Our implementation of this scheme for LIS has shown excellent load-balancing, scalability and optimal resource utilization. Our

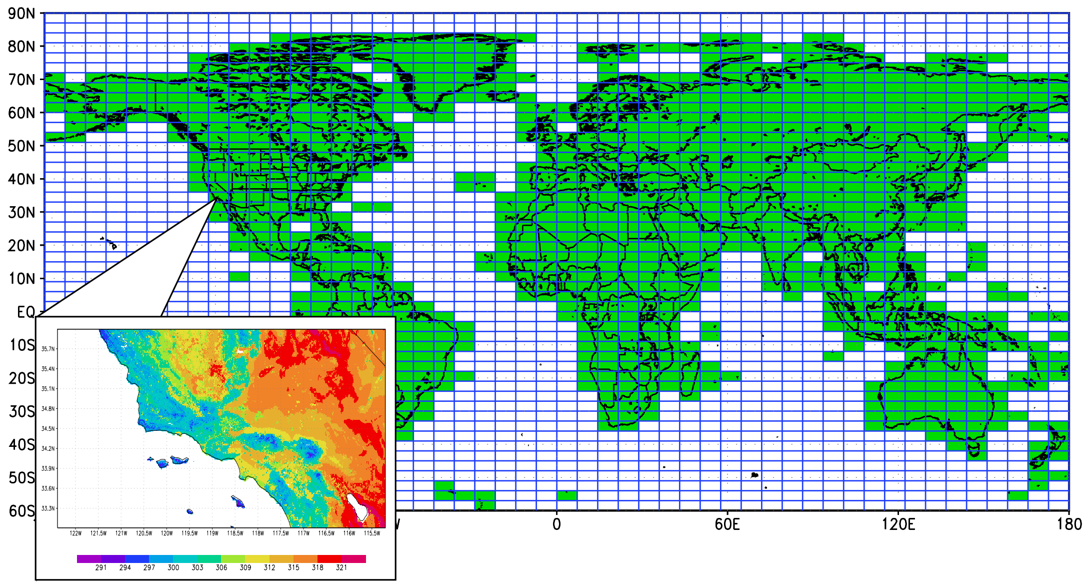

Fig. 3. Job partition of the global land surface for 1-km simulation. The global domain has 540×10^6 (36000×15000) grid points in total, which are partitioned into 2500 (50×50) blocks, with each block containing 216,000 (720×300) grid points. Those blocks that contain all ocean grid points are discarded, leaving a total of 1183 land blocks (green boxes). Each individual block (*e.g.*, the insert box) can be computed independently.

implementation also distinguishes itself with strong fault-tolerance in the face of node crashes from software or hardware failures, so that the job processing will not be interrupted even when a substantial number of the compute nodes crash during the run.

Figure 4 shows LIS' parallelization scheme. The "pool of tasks" paradigm we are using is equivalent to a master–slave programming model, where we use one of the I/O nodes as the master node and it distributes jobs to the slave (compute) nodes. We refer to the master node as "farmer" and the compute nodes as "dogs", and the jobs the master node gives out "bones".

The master node ("farmer") will keep three pools on hand when starting the job: pool of unfinished jobs ("bones"), finished ones ("done"), and ones fetched and being processes by compute nodes ("munching"). At the beginning, all the jobs are stored in the "bones" pool. Each compute node ("dog") sends a request to the master to request a job from it, and starts working on it when a job is assigned by the master node. The compute nodes do not get jobs directly by themselves from the "bones" pool, to eliminate the complexity of race condition management. The master node then moves the assigned jobs to the "munching" pool, and starts a timer for each assigned job. The timer specification will be based on the estimation of the execution time a compute node needs to finish a job. When a compute node finishes a job and notifies the master node before the job's corresponding timer runs out, this piece is regarded a finished job, and the master node moves it from the "munching" pool to the "done" pool. And the compute node goes on to request another job until the "bones" pool is empty.

The fault–tolerance is realized with the time–out mechanism based on the timer the master node keeps. If the timer of a fetched job runs out before the compute node reports back, the master node then assumes that that

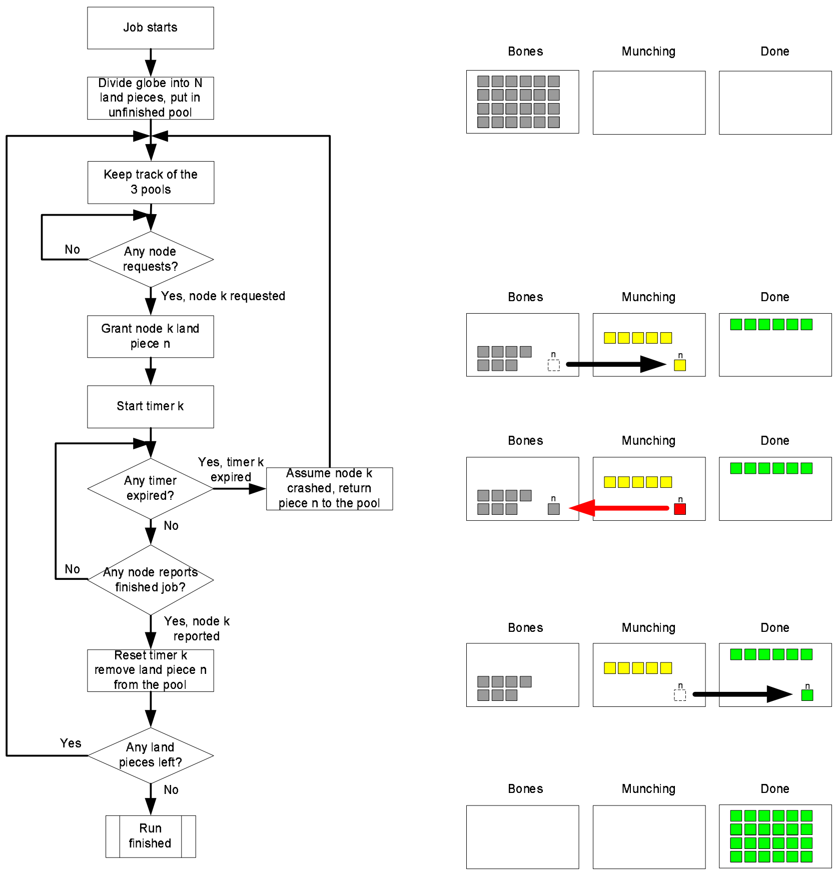
Job starts
Divide globe into N land pieces, put in unfinished pool
Keep track of the 3 pools
Any node requests?
No
Yes, node k requested
Grant node k land piece n
Start timer k
Any timer expired?
Yes, timer k expired
Assume node k crashed, return piece n to the pool
No
Any node reports finished job?
No
Yes, node k reported
Reset timer k remove land piece n from the pool
Any land pieces left?
Yes
No
Run finished
Bones
Munching
Done
n

Fig. 4. Task-pool-based parallelization scheme in LIS. Left panel shows the logic of the master node. Right panel shows the movement of jobs in the three pools: bones, munching and done. Each job in the "munching" pool is timed so when a job is timed out (red block), it will be put back to the "bones" pool for other nodes to handle. Time-out happens only when a compute node crashes. On the other hand, code crashes will be detected and handled immediately.

particular compute node must have crashed, and then moves that timed-out job from the "munching" table back to the "bones" table for other compute nodes to fetch. The flow-chart on the left of Fig. 4 shows the master node's job handling and scheduling process, and the various states of the three pools (right) corresponding to the stages in the flow-chart.

This farmer-dog system maximizes resource utilization even when the compute nodes ("dogs") have heterogeneous memory/CPU configuration, as the case with the LIS cluster. Each job ("bone") naturally has different number of total land points, depending on its location on the Earth. The job management system will make sure all the nodes will get to work on the biggest jobs they can handle first, depending on their hardware configuration. Only after they finish the biggest jobs can they continue to work on smaller jobs. This scheme ensures the more powerful nodes are not fighting for small jobs with less powerful nodes before they finish the jobs only they can handle. This has proven to be working well on the LIS cluster, since about half of the computer nodes have 1GB memory while the other half have only 512 MB.

Figure 5 shows one of the scaling test results for LIS with the job management system. The results were obtained

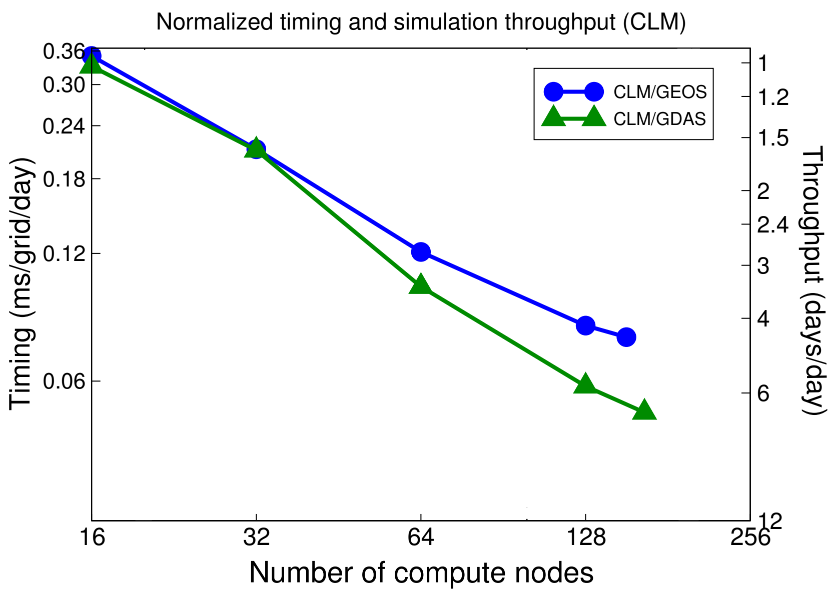

Fig. 5. Normalized timing (left y-axis) and simulation throughput (right y-axis) for LIS with one of the LSMs, CLM, and the two base forcing settings, GEOS and GDAS. Five (5) GDS servers (see Section V) were used. The maximum number of compute nodes used for CLM/GEOS is 152, and for CLM/GDAS 164.

by timing LIS runs with CLM for a 1-day simulation, and with the model forcing input from either the Global Data Assimilation System (GDAS) or the Goddard Earth Observing System (GEOS). The figure shows both wall-clock timing normalized to the number of grid points (left y-axis), and the corresponding normalized simulation throughput in terms of simulation days per wall-clock day (right y-axis).

As the figure shows, both runs showed excellent scalability. CLM/GEOS with 16 nodes ran at approximately $0.35ms/grid \cdot day$, which is approximately 1 simulation day per wall-clock day. With 32 nodes, CLM/GEOS timed $0.2ms/grid \cdot day$, or 1.7 days/day. Normalized timing kept decreasing nearly linearly with 64, 128 and 152 nodes, and with 152 nodes, we can simulate nearly 5 days in one day, which far exceeded performance needed for real-time simulation (1 day per day). The scaling for CLM/GDAS is similar, with an overall faster timing than CLM/GEOS, largely due to the more efficient feeding of the input GDAS data.

V. Parallel I/O and Distributed Data Storage

As we are using the large number of CPUs for the highly parallel runs for LIS, the input/output quickly became the performance bottleneck. First of all, while LIS is performing global land surface simulations with an ensemble of land surface models (*e.g.*, CLM, Noah and VIC), it requires a variety of input data, including land surface parameters and model and observational forcing. These input data need to be fed to each individual compute node dynamically with their desired subsets, depending on the location of the assigned block (Fig. 3). Next, after getting the input data, each compute node will produce output data with a volume of at least two orders more than the input data.

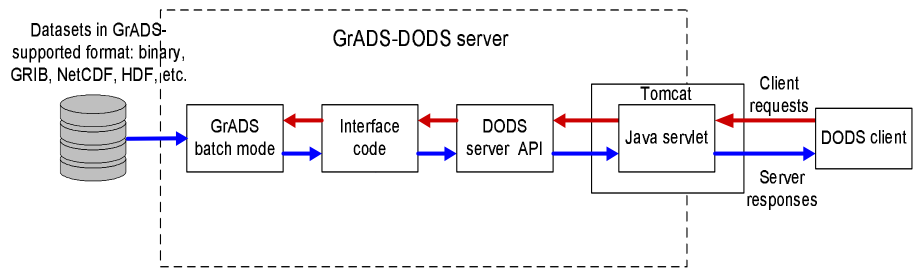

Fig. 6. The architecture of the GrADS-DODS (GDS) server. GDS uses Java Servlets to receive and interpret clients' data requests based on the HTTP-based DODS protocol. The client data requests are translated into DODS server API calls, which in turn, will invoke GrADS in batch mode. GrADS will perform the disk I/O to locate and retrieve the datasets with necessary subsetting and reformatting operations. GDS sends the retrieved data to the DODS client following the DODS protocol.

Figure 7 shows LIS' parallel I/O design and distributed storage, to solve the I/O bottleneck problem, and to make the I/O performance more scalable.

For input data serving, the GrADS-DODS (GDS) server played a key role. The GrADS-DODS server (Fig. 6) is an implementation of the DODS (Distributed Oceanographic Data System) protocol (http://www.unidata.ucar.edu/packages/dods/), with GrADS as the server backend to take advantage of GrADS' existing capability of data retrieving, subsetting and analysis. The GrADS-DODS server will provide data access via DODS protocol to DODS clients. The GrADS-DODS server uses a typical client-server architecture to communicate with the DODS clients. The communication protocol between a client and a server is based on HTTP. A GrADS-DODS server has the following components: Java servlets contained in the Tomcat servlet container, to handle the client requests and server replies via HTTP protocol; DODS server APIs, to parse the DODS requests and package output data; interface code, to translate the DODS requests into GrADS calls; and finally, GrADS running in batch mode, to actually process the requests, and perform data-retrieving, subsetting and processing on the server side.

We took advantage of the GrADS-DODS (GDS) server's dynamic sub-setting capability to serve only the subset of the input data to each compute node, thus reducing the total data traffic. To further boost input data serving throughput, we used multiple GDS servers running in parallel on the I/O nodes, with mirrored input data on the local storage of each I/O node (Fig. 6). We implemented a dynamic load balancing scheme to optimize the collective performance of the GDS servers.

To demonstrate how this setting affects LIS performance, we performed tests with 2, 3, 4, 5, and 6 GDS servers (I/O nodes). The timing results for Noah/GEOS runs with 128 and 180 compute nodes are shown in Fig. 8A.

As the figure shows, the performance improved as the number of GDS servers was increased. With 128 nodes and 2 GDS servers, Noah/GEOS ran at 0.122 ms/grid/day, while with 6 GDS servers, it ran at approximately 0.078 ms/grid/day, with a throughput improvement from 2.8 days/day to 4.4 days/day. However, the most significant performance improvement took place when the number of GDS servers was increased from 2 to 3. More GDS

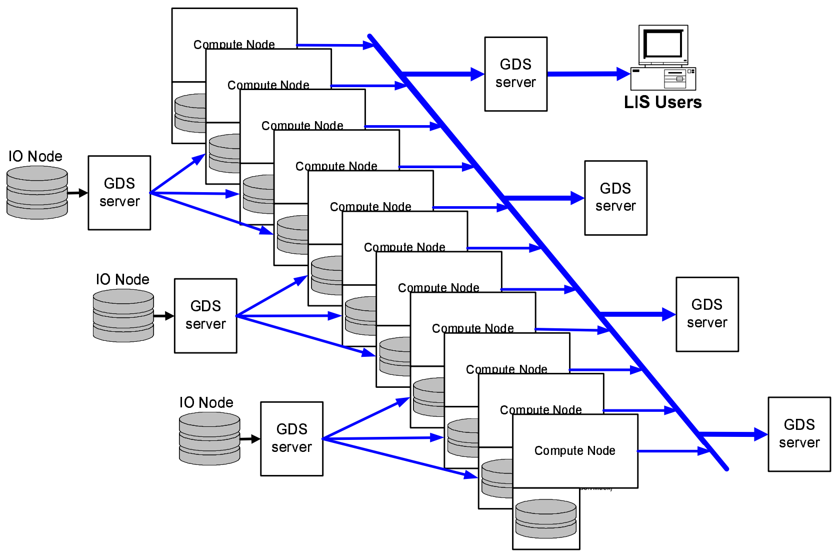

Fig. 7. Parallel I/O and distributed storage design for LIS. Input data are fed to the cluster by an array of parallel GDS servers (left). The compute nodes write their output data directly on their local disks. A modified GDS (right) can serve the output data from the compute nodes' disks, eliminating the need of data aggregation onto I/O nodes.

servers helped, but not as much. On the other hand, with 180 compute nodes, 2 and 3 GDS servers were certainly not enough, as we did not see significant performance increase compared to 128-node runs, due to the input data bottleneck with the small number of GDS servers. However, with 4,5 and 6 GDS servers, Noah runs with 180 nodes had a much better performance gain than the 128-node runs, with the most performance boost taking place with 4 GDS servers.

The output bottleneck posed a much more severe challenge if we were to follow the conventional approach to first aggregate the output data to a central storage system and then serve the users from there. We completely eliminated the need for central data storage by modifying the GDS server system, to expand its capability to serve data directly off the local disks of every compute nodes. Therefore, each compute node directly writes its output data to its local disk, and then the modified GDS system can treat the collection of the compute node disks as a big disk system, and serve the output data to users from there the same way as from a single local disk.

By implementing this scheme, we gained two critical performance benefits. The first is that the output bottleneck for LIS runs is completely eliminated, when each compute node writes the output data to its local disk, eliminating the data aggregation traffic. Otherwise, it is estimated that it would take 4 to 10 days to finish a 1 day LIS simulation, due to the traffic congestion and limited throughput of such a central scheme.

The second benefit is the speed-up of serving the output data to users. The enhanced GDS server can gather the output in parallel from the compute nodes, resulting in much higher data throughput than from a single local disk. Figure 8B shows the performance comparison for serving users off the cluster storage versus from the local storage. To retrieve a subset of the size of continental U.S. from the global output data, it took 9.4 seconds when the output data is aggregated and stored on a single local disk, while it took 0.8 seconds from the distributed cluster storage.

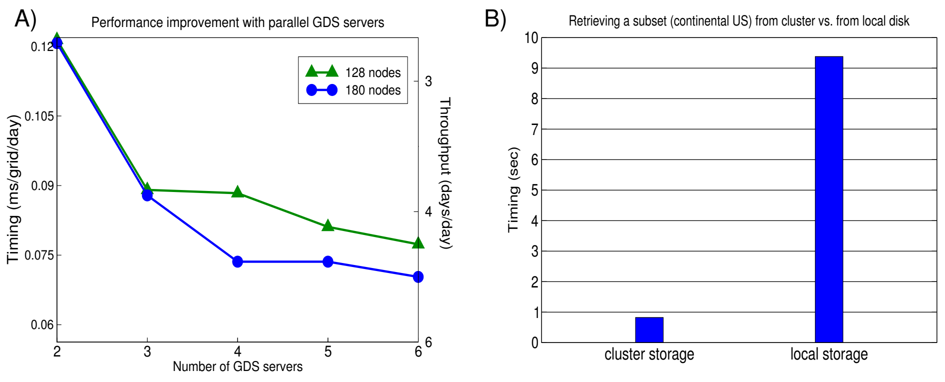

Fig. 8. A: Performance improvement with parallel GDS servers for input data serving, as shown in Fig. 7. Normalized timing (left y-axis) and simulation throughput (right y-axis) are shown functions of the number of GDS servers used, for LIS with Noah model, GEOS base forcing and with 128 and 180 compute nodes, respectively. Two (2) to 6 GDS servers were used, with a dynamic load balancing mechanism. B: Performance improvement for retrieving LIS output data from the parallel cluster-based storage implementation shown in Fig. 7 (left bar), compared with the same retrieval but from local disk storage on an I/O node (right bar).

This implementation shares some similarities with the Parallel Virtual File System (PVFS, [1]), in that the data are stored distributedly and I/O operations are performed in parallel. However, our storage scheme does not do a fine-grained data striping across all the storage nodes; instead, the data output is directly written to the local disk without incurring network traffic to other nodes. Therefore the output scales linearly to the number of nodes, and in case of node failures, the I/O on other working nodes is not affected. This scheme works particularly well with geoscience data when the data can be partitioned spatially into adjacent blocks, while PVFS is a more generic and extensive parallel file system with all the standard UNIX I/O semantics.

VI. Data Replication with Peer-to-Peer Technology

Although it is trivial to copy small files from one computer to a few other computers over the network via conventional approaches such as NFS, FTP or HTTP based file sharing, it becomes a formidable task to copy a large amount of data from one computer to hundreds of other computers by this means, as with the case for data replication on the LIS Beowulf cluster. The traditional client–server paradigm, such as NFS, won't scale when tens, or hundreds of clients are copying data from the server, as either the server's bandwidth will be quickly saturated or the server gets overloaded with degraded performance. In addition, the chance of data corruption greatly increases as the data volume and network traffic are increases. Worse still, if a node crashes in the middle of data replication, it has to start over again from the beginning when it comes back online.

LIS needs to pre-stage several Gigabytes of data to every compute node from one node, as these data are static parameters and shared by all the compute nodes, and it would be too expensive to let compute nodes to access the data during runtime over the network. To efficiently replicate these datasets within the cluster without incurring the complexity of client–server approaches, we introduced one of the P2P technologies, BitTorrent (BT), to serve as the data replication channel on the cluster. Originally designed for sharing big music or video files by Web surfers,

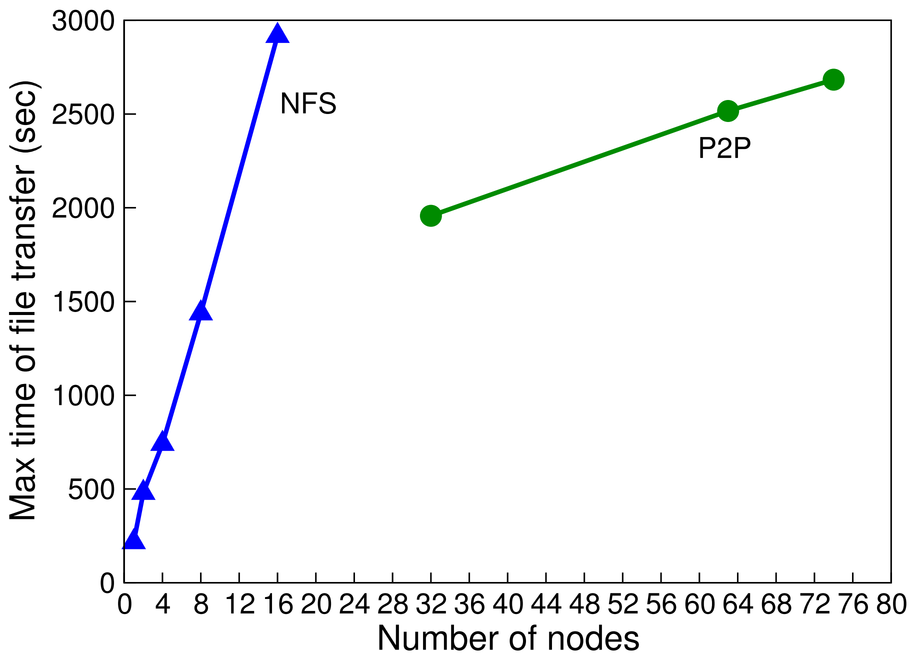

Fig. 9. Comparison of conventional NFS-based and P2P-based data replication performance, measured by the maximum time needed to copy a test file to different numbers of compute nodes. The test file contains 2GB random data. All the nodes were inter-connected with fast Ethernet links.

BT's serverless feature and mechanisms to handle big files reliably will be a perfect fit for the particular LIS data replication situation.

To get data from one computer (the "seed") to many others ("peers"), BT first splits the data on the seed into small chunks. Then each peer get a random chunk from the seed. Then the peers can exchange with one another of the chunks and eventually every peer will get all the chunks, thus the whole dataset. Thus the data transfer is mostly taking place between the peers. There is a meta-data server ("tracker") which keeps track which peer has which chunks, and also records the message digest of each chunk. With the message digest, each peer can guarantee the integrity of each chunk, eliminating the possibility of data corruption. Finally, each peer can resume previous transfer without starting over after a crash.

Figure 9 shows the performance comparison for data replication on the LIS cluster, between our new P2P data sharing and the conventional NFS-based file copying. With NFS, the time needed to replicate the test file increased quickly when the number of nodes was increased, as NFS quickly reached the physical limit of the server's bandwidth with less then 10 clients. It will take nearly 3000 seconds to copy the data file to 16 nodes. With BitTorrent, however, the time increased much more slowly with the number of nodes, and it even took less time to replicate the file to 74 nodes than to 16 nodes with NFS. Clearly the P2P-based replication scaled much better.

VII. Summary

The realtime, high-performance requirement of LIS poses great challenges in the computing technology today. We have to take non-conventional and innovative approaches to meet these challenges and to keep pushing the envelope. Our contribution in the advanced computational technologies made LIS a successful and high-performance Earth modeling system.

We built a low-cost Beowulf cluster from commodity components to support LIS' intensive computing needs. To fully utilize the computing power of the cluster effectively, and to take advantage of the parallel nature of land surface simulations, we developed our own job management system, which demonstrated great flexibility and strong fault-tolerance, and plays a crucial rule in assuring LIS to run most efficiently and fully utilizing the cluster.

With LIS' scale of computation, the I/O is the greatest potential performance bottleneck. We abandoned the traditional approach of output aggregation on central storage devices, and enhanced GDS server with innovative design to let it serve distributedly-stored data to users, and set up the first GDS server farm with parallel servers and dynamic load balancing, to greatly increase the input performance of the whole system.

The need of large scale data replication for LIS' computational needs promoted us to turn to novel approaches. We applied P2P technology to help meet the performance goal of high volume data replication with a large number of nodes. Our implementation of BitTorrent on the Linux cluster proved to be highly effective with our data replication needs.

With the performance results demonstrated by LIS, and with the practical experience we gained from exploring and employing new computational technologies, we believe these technologies will be helpful in other areas as well.

Acknowledgment

We greatly appreciate the assistance from Susan Olden, Luther Lighty, Joe Wielgosz, Kevin Miller, Meg Larko, Aedan Jenkins, Uttam Majumder, Nikkia Anderson and Steve Lidard. LIS is a Grand Challenge Investigation funded under NASA ESTO/CT CAN-00-OES-01, with additional support from NASA ESTO/AIST NRA-02-OES-04.

References

[1] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. Pvfs: A parallel file system for linux clusters. *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA*, pages 317–327, 2000.

[2] Y. Dai, X. Zeng, R. E. Dickinson, I. Baker, G. B. Bonan, and M. G. Bosilovich. The common land model. *Bull. Amer. Meteor. Soc.*, 84:1013–1023, 2003.

[3] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[4] C. Hill, C. Deluca, V. Balaj, M.Suarez, and A. da Silva. The architecture of the earth system modeling framework. *Computing in Science & Engineering*, 6:18–28, 2004.

[5] H. P. Hofstee, J. J. Likkien, and J. L. A. Van De Snepscheut. A distributed implementation of a task pool. *Research Directions in High-Level Parallel Programming Languages*, pages 338–348, 1991.

[6] C. D. Peters-Lidard, S. Kumar, Y. Tian, J. L. Eastman, and P. Houser. Global urban-scale land-atmosphere modeling with the land information system. *Symposium on Planning, Nowcasting, and Forecasting in the Urban Zone, 84th AMS Annual Meeting 11-15 January 2004 Seattle, WA, USA.*, 2004.

[7] E. W. Sanderson, M. Jaiteh, M. A. Levy, K. H. Redford, A. V. Wannebo, and G. Woolmer. The human footprint and the last of the wild. *Bioscience*, 52:891–904, 2002.